(12) **INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)**

[Continued on next page]

(54) **Title: READ LOCK MISS CONTROL IN A MULTITHREADED ENVIRONMENT**

(57) **Abstract:** Managing memory access to random access memory includes fetching a read lock memory reference request and placing the read lock memory reference request at the end of a read lock miss queue if the read lock memory reference request is requesting access to an unlocked memory location and the read lock miss queue contains at least one read lock memory reference request.

patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *Without international search report and to be republished upon receipt of that report.*

READ LOCK MISS CONTROL IN A MULTITHREADED ENVIRONMENT

BACKGROUND OF THE INVENTION

This invention relates to read lock memory references.

When a computer instruction (thread) needs access to a

5   memory location in the computer, it goes to a memory

controller. A memory reference instruction may request a

read lock on a particular memory location. The read lock

prevents other instructions from accessing that memory

location until a read unlock instruction for that memory

10   location gets granted.

SUMMARY OF THE INVENTION

According to one aspect of the invention, a method is

described of managing memory access to random access memory

includes fetching a read lock memory reference request and

15   placing the read lock memory reference request at the end

of a read lock miss queue if the read lock memory reference

request is requesting access to an unlocked memory location

- 1 -

and the read lock miss queue contains at least one read

lock memory reference request.

Other advantages will become apparent from the

following description and from the claims.

5   BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a communication system

employing a hardware-based multithreaded processor.

FIG. 2 is a block diagram of a memory controller for

latency limited operations used in the hardware-based

10   multithreaded processor.

FIG. 3 is a flowchart of the operation of a memory

controller in the hardware-based multithreaded processor.

FIG. 4 is a flowchart of the operation of a memory

controller in the hardware-based multithreaded processor.

15   DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, a communication system 10

includes a parallel, hardware-based multithreaded processor

12. The hardware-based multithreaded processor 12 is

coupled to a bus such as a peripheral component

- 2 -

interconnect (PCI) bus 14, a memory system 16, and a second

bus 18. The system 10 is especially useful for tasks that

can be broken into parallel subtasks or functions.

Specifically, a hardware-based multithreaded processor 12

5   is useful for tasks that are bandwidth oriented rather than

latency oriented. The hardware-based multithreaded

processor 12 has multiple microengines 22a-22f, each with

multiple hardware controlled threads that can be

simultaneously active and independently work on a task.

10   The hardware-based multithreaded processor 12 also

includes a central controller 20 that assists in loading

microcode control for other resources of the hardware-based

multithreaded processor 12 and performs other general

purpose computer type functions such as handling protocols,

15   exceptions, and extra support for packet processing where

the microengines 22a-22f pass the packets off for more

detailed processing such as in boundary conditions. In one

embodiment, the processor 20 is a Strong Arm® (Arm is a

trademark of ARM Limited, United Kingdom) based

20   architecture. The general purpose microprocessor 20 has an

operating system. Through the operating system the

processor 20 can call functions to operate on microengines

22a-22f. The processor 20 can use any supported operating

- 3 -

system, preferably a real time operating system. For the

core processor implemented as a Strong Arm architecture,

operating systems such as, MicrosoftNT real-time, VXWorks

and μCUS, a freeware operating system available over the

5   Internet, can be used.

The hardware-based multithreaded processor 12 also

includes a plurality of function microengines 22a-22f.

Functional microengines (microengines) 22a-22f each

maintain a plurality of program counters in hardware and

10   states associated with the program counters. Effectively,

a corresponding plurality of sets of threads can be

simultaneously active on each of the microengines 22a-22f

while only one is actually operating at any one time.

In one embodiment, there are six microengines 22a-22f

15   as shown. Each microengines 22a-22f has capabilities for

processing four hardware threads. The six microengines

22a-22f operate with shared resources including memory

system 16 and bus interfaces 24 and 28. The memory system

16 includes a Synchronous Dynamic Random Access Memory

20   (SDRAM) controller 26a and a Static Random Access Memory

(SRAM) controller 26b. SDRAM 16a and SDRAM controller 26a

are typically used for processing large volumes of data,

e.g., processing of network payloads from network packets.

SRAM 16b and SRAM controller 26b are used in a networking

implementation for low latency, fast access tasks, e.g.,

accessing look-up tables, memory for the core processor 20,

and so forth.

5   The six microengines 22a-22f access either the SDRAM

16a or SRAM 16b based on characteristics of the data.

Thus, low latency, low bandwidth data is stored in and

fetched from SRAM 16b, whereas higher bandwidth data for

which latency is not as important, is stored in and fetched

10   from SDRAM 16a. The microengines 22a-22f can execute

memory reference instructions to either the SDRAM

controller 26a or the SRAM controller 26b.

Advantages of hardware multithreading can be explained

by SRAM or SDRAM memory accesses. As an example, an SRAM

15   access requested by a Thread_0, from a microengine 22a-22f

will cause the SRAM controller 26b to initiate an access to

the SRAM 16b. The SRAM controller 26b controls arbitration

for the SRAM bus 27, accesses the SRAM 16b, fetches the

data from the SRAM 16b, and returns data to the requesting

20   microengine 22a-22f. During an SRAM 16b access, if the

microengine, e.g., 22a, had only a single thread that could

operate, that microengine would be dormant until data was

returned from the SRAM 16b. The hardware context swapping

- 5 -

within each of the microengines 22a-22f enables other

contexts with unique program counters to execute in that

same microengine. Thus, another thread, e.g., Thread_1,

can function while the first thread, e.g., Thread_0, is

5   awaiting the read data to return. During execution,

Thread_1 may access the SDRAM memory 16a. While Thread_1

operates on the SDRAM unit 26a, and Thread_0 is operating

on the SRAM unit 26b, a new thread, e.g., Thread_2, can now

operate in the microengine 22a. Thread_2 can operate for a

10   certain amount of time until it needs to access memory or

perform some other long latency operation, such as making

an access to a bus interface. Therefore, simultaneously,

the processor 12 can have a bus operation, SRAM operation,

and SDRAM operation all being completed or operated upon by

15   one microengine 22a and have one more thread available to

process more work in the data path.

The hardware context swapping also synchronizes

completion of tasks. For example, two threads could hit

the same shared resource, e.g., SRAM 16b. Each one of

20   these separate functional units, e.g., the FBUS interface

28, the SRAM controller 26b, and the SDRAM controller 26a,

when they complete a requested task from one of the

microengine thread contexts reports back a flag signaling

- 6 -

completion of an operation. When the flag is received by
the microengine 22a-22f, the microengine 22a-22f can
determine which thread to turn on.

One example of an application for the hardware-based
multithreaded processor 12 is as a network processor. As a
network processor, the hardware-based multithreaded
processor 12 interfaces to network devices such as a media
access controller device, e.g., a 10/100BaseT Octal MAC 13a
or a Gigabit Ethernet device 13b. In general, as a network
processor, the hardware-based multithreaded processor 12
can interface to any type of communication device or
interface that receives/sends large amounts of data. If
communication system 10 functions in a networking
application, it could receive a plurality of network
packets from the devices 13a, 13b and process those packets
in a parallel manner. With the hardware-based
multithreaded processor 12, each network packet can be
independently processed.

Another example for use of processor 12 is a print
engine for a postscript processor or as a processor for a
storage subsystem, i.e., RAID disk storage. A further use
is as a matching engine. In the securities industry for
example, the advent of electronic trading requires the use

- 7 -

of electronic matching engines to match orders between
buyers and sellers. These and other parallel types of
tasks can be accomplished on the system 10.

Each of the functional units, e.g., the FBUS interface
5   28, the SRAM controller 26b, and the SDRAM controller 26a,
are coupled to one or more internal buses. The internal
buses are dual, 32-bit buses (i.e., one bus for read and
one for write). The hardware-based multithreaded processor
12 also is constructed such that the sum of the bandwidths
10 of the internal buses in the processor 12 exceeds the
bandwidth of external buses coupled to the processor 12.
The processor 12 includes an internal core processor bus
32, e.g., an ASB bus (Advanced System Bus), that couples
the processor core 20 to the memory controller 26a, 26b and
15 to an ASB translator 30. The ASB bus 32 is a subset of the
so-called AMBA bus that is used with the Strong Arm
processor core. The processor 12 also includes a private
bus 34 that couples the microengine units to SRAM
controller 26b, ASB translator 30, and FBUS interface 28.
20 A memory bus 38 couples the memory controllers 26a, 26b to
the bus interfaces 24 and 28 and memory system 16 including
a flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, the SRAM controller 26b for the

SRAM 16b is shown. The SRAM controller 26b includes an

address and command queue 120. SRAM controller 26b is

optimized based on the type of memory operation, i.e., a

5   read or a write. The address and command queue 120

includes a high priority queue 120a (holding memory

reference requests from high priority tasks), a read queue

120b (holding read memory reference requests, which are the

predominant memory reference functions that an SRAM

10   performs), and an order queue 120c (holding, in general,

all writes to SRAM 16b and reads that are to be non-

optimized). Although not shown, the address and command

queue 120 could also include a write queue. An insert

queue control unit 132 of control logic determines where to

15   queue each memory request from the microengines 22a-22f.

An arbitration scheme in a remove queue arbitration unit

124 of control logic determines the processing order of the

memory reference requests in the queues 120.

The SRAM controller 26b also includes core bus

20   interface logic, e.g., ASB bus 122. The ASB bus interface

logic 122 interfaces the core processor 20 to the SRAM

controller 26b. The ASB bus 122 is a bus that includes a

32-bit data path and a 28-bit address path. The data is

- 9 -

accessed to and from memory 16b through an MEM

(microelectromechanical) ASB data device 128, e.g., a

buffer. The MEM ASB data device 128 is a queue for write

data. If there is incoming data from the core processor 20

5   via ASB interface 122, the data can be stored in the MEM

ASB data device 128 and subsequently removed from the MEM

ASB data device 128 through an SRAM interface 140 to SRAM

memory 16b. Although not shown, the same queue structure

can be provided for reads.

10   The memory requests are sent to SRAM interface 140 via

a multiplexer 126. The multiplexer 126 is controlled by an

SRAM arbiter 131 which detects the fullness of each of the

queues and the status of the requests and from that decides

priority based on a programmable value stored in a priority

15   service control register 130. Once control to the

multiplexer 126 selects a memory reference request, the

memory reference request is sent to a command decoder and

address generator 138 where it is decoded and an address is

generated.

20   The SRAM controller 26b maintains control of the

Memory Mapped off-chip SRAM and Expansion ROM. The SRAM

controller 26b can address, e.g., 16 MBytes, with, e.g., 8

MBytes mapped for SRAM 16b, and 8 MBytes reserved for

special functions including: Boot space via flashrom 16c;

and Console port access for MAC devices 13a, 13b and access

to associated (RMON) counters. The SRAM is used for local

look-up tables and queue management functions.

5   The SRAM controller 26b supports the following

transactions:

          Microengine requests   (via private bus) to/from SRAM.
          Core Processor   (via ASB bus)   to/from SRAM.

10

          The SRAM controller 26b performs memory reference

sorting to minimize delays (bubbles) in the pipeline from

the SRAM interface 140 to memory 16b. The SRAM controller

26b does memory reference sorting based on the read

15 function. A bubble can either be one or two cycles

depending on the type of memory device employed.

          The SRAM controller 26b includes a lock look-up device

142, an eight-entry address content addressable memory

(CAM) for look-ups of read locks. Each position includes a

20 valid bit that is examined by subsequent read lock

requests. The address and command queue 120 also includes

a read lock miss (fail) queue 120d. The read lock miss

queue 120d is used to hold up to twenty-four read requests

- 11 -

that fail because of a lock existing on the portion of
memory requested to be read. One of the microengines 22a-
22f issues a read lock memory reference request that is
processed in address and control queue 120.

5   Referring to FIG. 3, a memory reference request is
processed 51 by the controller 26b. The controller 26b
receives 50 a memory reference request on either the order
queue 120c or the read queue 120b. The command decoder and
address generator 138 determines 52 whether the memory

10   reference request is a "read" request, a write request, a
"read lock" request, or a write unlock ("unlock") request.
If the command decoder and address generator 138 determines
52 that the request is a read request, then the command
decoder and address generator 138 enters 54 the lock into

15   the lock look-up device 142 so that subsequent read lock
requests will find the memory location locked. Lock
requests cannot be processed on that locked memory location
until an unlock request from a microengine 22a-22f unlocks
58 that memory location.

20   If the command decoder and address generator 138
determines 52 that the memory reference request is a read
lock request, the controller 26b will access 56 lock look-
up device 142 to determine whether this memory location is

- 12 -

already locked. The access 56 determines whether the read

lock request gets performed 62 or enters 64 the read lock

miss queue 120d where it waits for the memory location to

be unlocked.

5   If the memory location is locked from any prior lock

request, the read lock request fails and gets stored 64 at

the end of the read lock miss queue 120d. The controller

26b proceeds to receive 50 the next request.

       If the memory location is unlocked, the lock look-up

10   device 142 checks 60 to see if the read lock miss queue

120d is empty. If the read lock miss queue 120d is empty,

that indicates that there are no previous read lock

requests awaiting processing. Thus, the read lock request

will be processed: the address of the memory reference will

15   be used by the SRAM interface 140 to perform 62 an SRAM

address read/write request to memory 16b and subsequently

lock the memory location. Then the controller 26b receives

50 the next request.

       If the read lock miss queue 120d is not empty, that

20   indicates that there is at least one prior read lock

request awaiting processing. To ensure a first-in/first-

out order of processing the read lock requests, the read

lock request will be stored 64 at the end of the read lock

- 13 -

miss queue 120d. Read lock requests continue queuing in
the read lock miss queue 120d until the controller 26b
receives 50 an unlock request and recognizes 52 it as such.
A memory location is unlocked by operation of a

5   microcontrol instruction in a program after the need for
the lock has ended. Once the unlock request is received
50, the memory location is unlocked 58 by clearing the
valid bit in the CAM 142. After the unlock 58, the read
lock fail queue 120d becomes the highest priority queue

10   120, giving all queued read lock misses a chance to issue a
memory lock request.

        Referring to FIG. 4, the read lock miss queue 120d is
tested 69 to see if the head of the read lock miss queue
120d may be granted its lock. The testing 69 will continue

15   until the read lock miss queue 120d fails to provide a
grant or until there are no more entries in the read lock
miss queue 120d. The highest priority is given to the read
lock miss queue 120d by letting it win 70 the next
arbitration in the remove queue arbitration unit 124. The

20   lock look-up device 142 checks 72 to see if the top read
lock request (the one stored first in the read lock miss
queue 120d) is requesting to read an unlocked memory
location. If the read lock request is for a locked

- 14 -

location, the read lock miss queue 120d remains as it is to

ensure the first-in/first-out order described above, and

the testing 69 ends 78. If the read lock request is for an

unlocked location, the address of the memory reference will

5   be used by the SRAM interface 140 to perform 74 an SRAM

address read/write request to memory 16b. After performing

74, the lock look-up device 142 checks 76 to see if the

read lock miss queue 120d is empty. If it is empty, the

testing 69 ends 78. Otherwise, the testing 69 continues

10 with the read lock miss queue 120d winning 70 another

arbitration in the remove queue arbitration unit 124.

Other embodiments are within the scope of the

following claims.

What is claimed is:

1.   A method of managing memory access to random access memory comprises:

fetching a read lock memory reference request; and

placing the read lock memory reference request at the end of a read lock miss queue if the read lock memory reference request is requesting access to an unlocked memory location and the read lock miss queue contains at least one read lock memory reference request.

2.   The method of claim 1 in which the random access memory is located in a parallel, hardware-based multithreaded processor.

3.   The method of claim 1 in which the random access memory comprises a static random access memory.

4.   The method of claim 1 further comprises placing the read lock memory reference at the end of a read lock miss queue if the read lock memory reference request is requesting access to a locked memory location.

5.   The method of claim 1 further comprises removing the read lock memory reference requests from the read lock

miss queue in the order in which they were placed in the read lock miss queue.

6.   The method of claim 1 in which the read lock memory reference request is fetched from a read queue.

7.   The method of claim 1 in which the read lock memory reference request is fetched from an order queue.

8.   The method of claim 1 in which the read lock miss queue contains a number of entries equal to a number of contexts that may execute the read lock memory reference requests minus one.

9.   The method of claim 1 further comprises placing a read lock memory reference request in a queue that contains a read lock memory reference request requesting access to the same memory location.

10. An article comprising a computer-readable medium which stores computer-executable instructions for managing memory access to random access memory, the instructions causing a computer to:

    fetch a read lock memory reference request; and

    place the read lock memory reference request at the

- 17 -

end of a read lock miss queue if the read lock memory

reference request is requesting access to an unlocked

memory location and the read lock miss queue contains at

least one read lock memory reference request.

5   11.   The article of claim 10 in which the random

access memory is located in a parallel, hardware-based

multithreaded processor.

12.   The article of claim 10 in which the random

access memory comprises a static random access memory.

10   13.   The article of claim 10 in which the instructions

further cause a computer to place the read lock memory

reference request at the end of a read lock miss queue if

the read lock memory reference request is requesting access

to a locked memory location.

15   14.   The article of claim 10 in which the instructions

further cause a computer to remove the read lock memory

reference requests from the read lock miss queue in the

order in which they were placed in the read lock miss

queue.

15. The article of claim 10 in which the read lock memory reference request is fetched from a read queue.

16. The article of claim 10 in which the read lock memory reference request is fetched from an order queue.

5   17. The article of claim 10 in which the read lock miss queue contains a number of entries equal to a number of contexts that may execute the read lock memory reference requests minus one.

18. The article of claim 10 in which the instructions
10   further cause a computer to place a read lock memory reference request in a queue that contains a read lock memory reference request requesting access to the same memory location.

19. A controller for managing access to random access
15   memory comprises:
    an address content addressable memory that supports locked locations in memory; and
    a read lock miss queue that stores read lock memory reference requests that request access to memory locations
20   unlocked in the address content addressable memory when the read lock miss queue contains at least one entry.

- 19 -

20. The controller of claim 19 in which the random access memory is located in a parallel, hardware-based multithreaded processor.

21. The controller of claim 19 in which the random access memory comprises a static random access memory.

22. The controller of claim 19 in which the read lock memory reference requests are stored at the end of the read lock miss queue if the read lock memory reference request requests access to a locked memory location.

23. The controller of claim 19 in which the read lock memory reference requests are removed from the read lock miss queue in the order in which they were placed in the read lock miss queue.

24. The controller of claim 19 in which the read lock miss queue contains a number of entries equal to a number of contexts that may execute the read lock memory reference requests minus one.
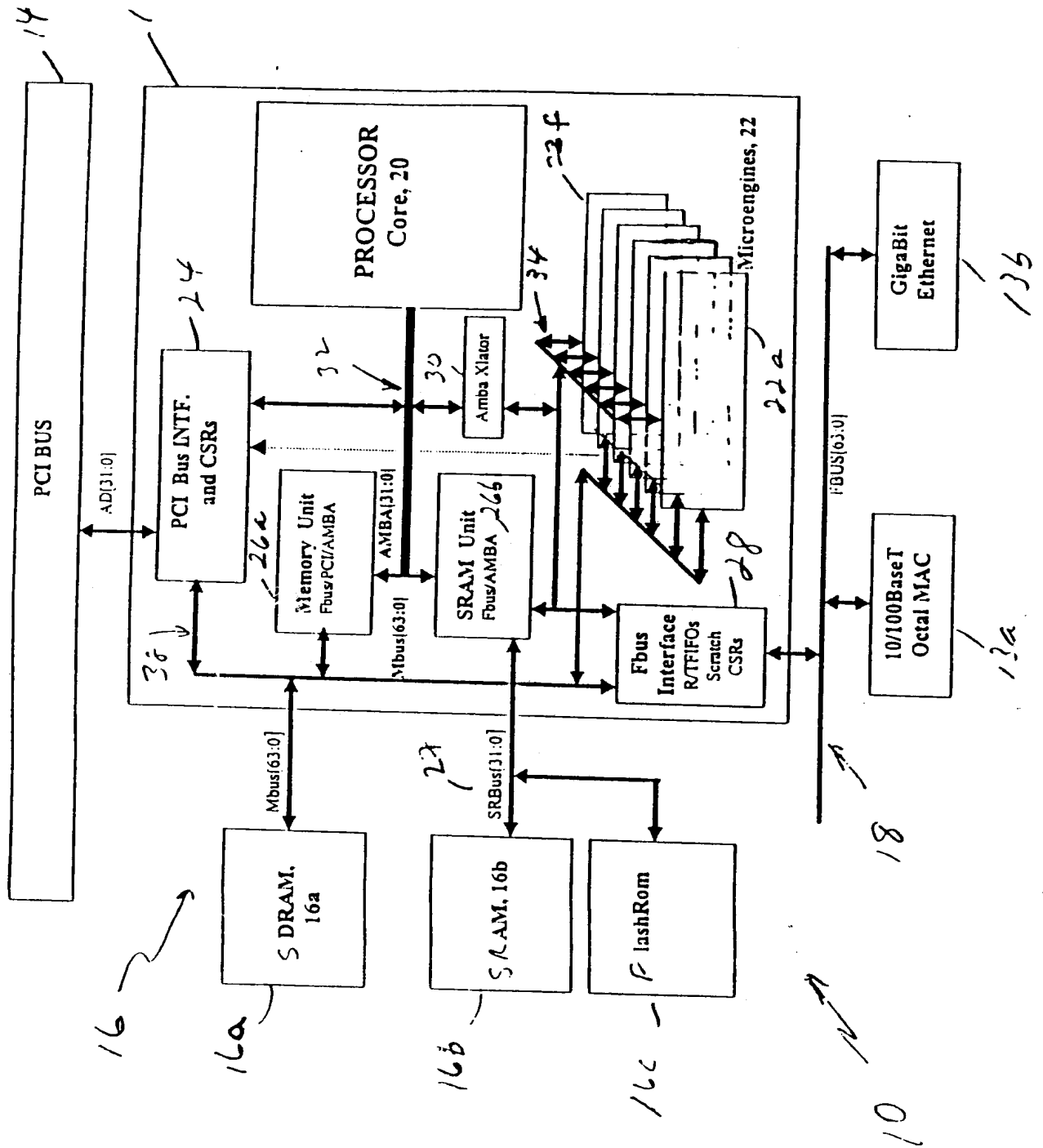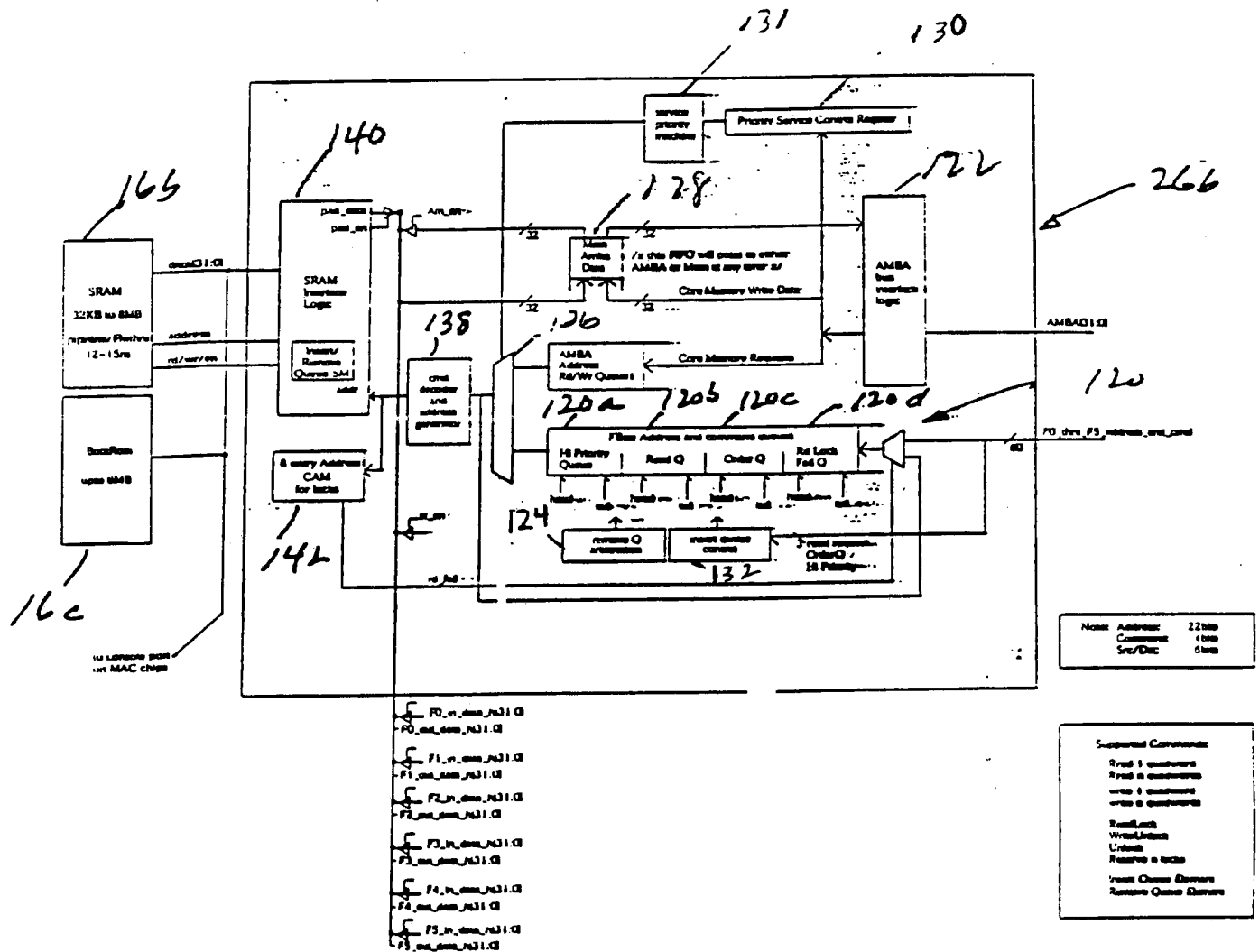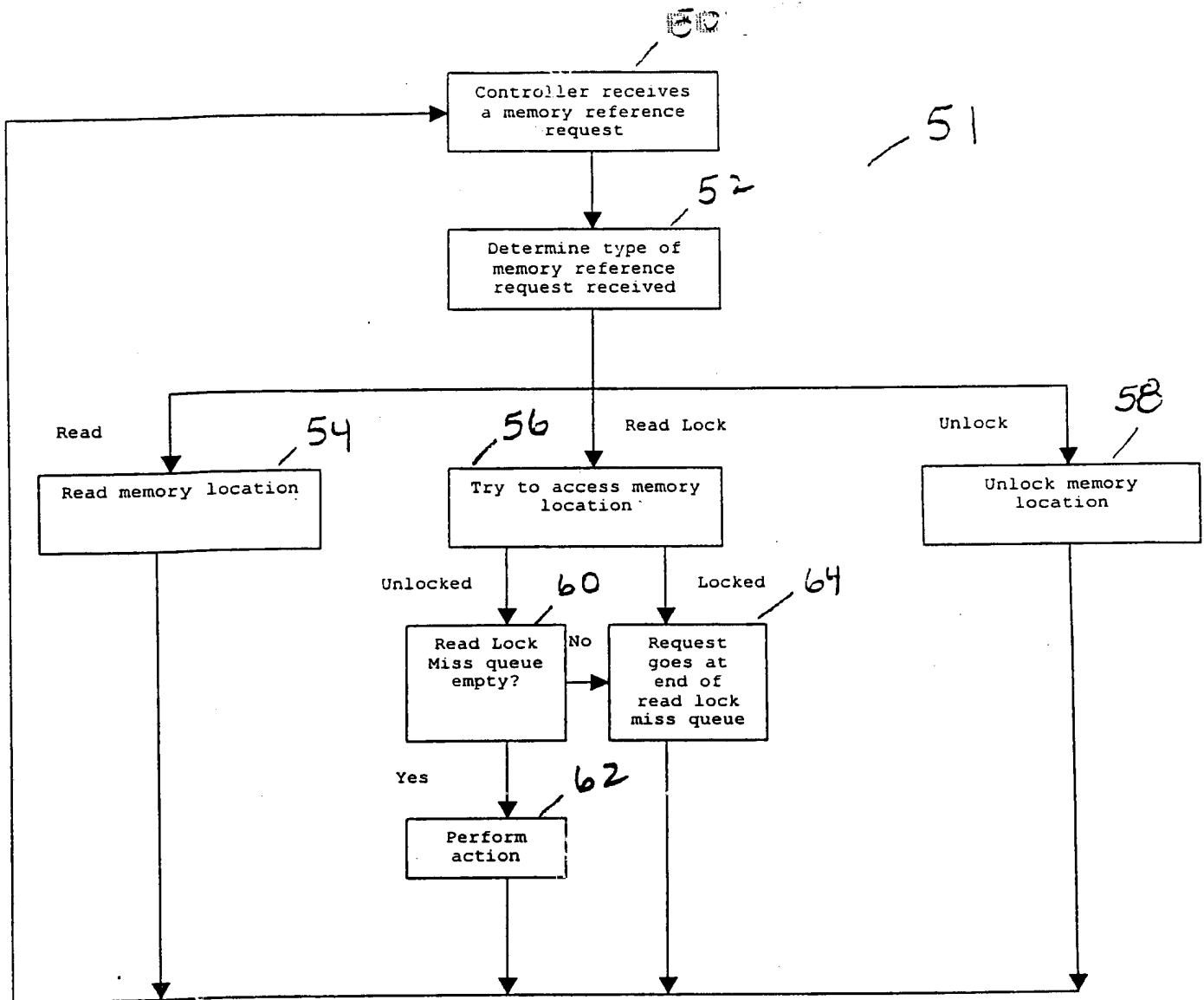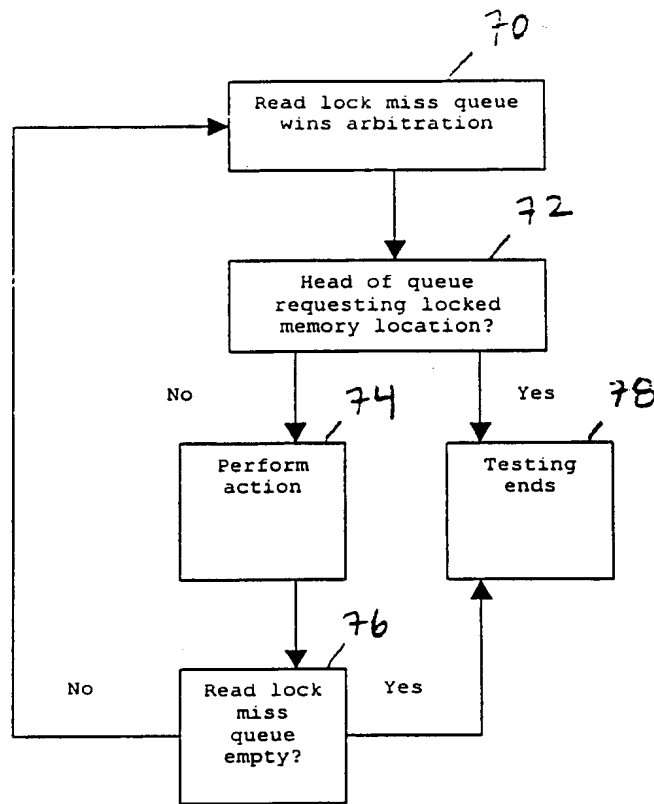
FIG.1

FIG. 2

FIG. 3

70

67

```
┌─────────────────────────┐
│   Read lock miss queue  │
│     wins arbitration    │
└─────────────────────────┘
```

72

```
┌─────────────────────────┐
│     Head of queue       │
│   requesting locked     │
│   memory location?      │
└─────────────────────────┘
```

No   74   Yes   78

```
┌───────────┐           ┌───────────┐
│  Perform  │           │  Testing  │
│  action   │           │   ends    │
│           │           │           │
└───────────┘           └───────────┘
```

76

No   Yes

```
┌───────────┐
│ Read lock │
│   miss    │
│  queue    │
│  empty?   │
└───────────┘
```

FIG. 4

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(54) Title: READ LOCK MISS CONTROL IN A MULTITHREADED ENVIRONMENT

(57) Abstract: Managing memory access to random access memory includes fetching a read lock memory reference request and placing the read lock memory reference request at the end of a read lock miss queue if the read lock memory reference request is requesting access to an unlocked memory location and the read lock miss queue contains at least one read lock memory reference request.

WO 01/48596 A3

NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

(88) **Date of publication of the international search report:**
21 March 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7   G06F9/52

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7   G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 464 715 A (DIGITAL EQUIPMENT CORP) 8 January 1992 (1992-01-08) column 7, line 31 – line 42 column 10, line 55 –column 11, line 39 column 12, line 7 –column 13, line 40 column 14, line 6 – line 41 ---	1-24
A	US 5 517 648 A (SANFACON MARC ET AL) 14 May 1996 (1996-05-14) column 11, line 21 – line 53 column 14, line 25 – line 41 column 18, line 14 –column 19, line 15 --- -/--	2,3,6,7, 11,12, 15,16, 20,21

| [X] | Further documents are listed in the continuation of box C. | [X] | Patent family members are listed in annex. |

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 November 2001	16/11/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Carciofi, A

Form PCT/ISA/210 (second sheet) (July 1992)

1

page 1 of 2

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 140 685 A (SIPPLE RALPH E ET AL) 18 August 1992 (1992-08-18) column 4, line 54 -column 5, line 40 column 13, line 32 -column 14, line 2 -----	1,4,5, 10,13, 14,19, 22,23

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0464715	A	08-01-1992	AU	633724 B2	04-02-1993
			AU	7527991 A	12-03-1992
			CA	2042772 A1	30-12-1991
			DE	69128815 D1	05-03-1998
			DE	69128815 T2	27-08-1998
			EP	0464715 A2	08-01-1992
			US	5341491 A	23-08-1994
US 5517648	A	14-05-1996	US	6311286 B1	30-10-2001
			US	5956522 A	21-09-1999
			US	6125436 A	26-09-2000
			US	5809340 A	15-09-1998
			US	5522069 A	28-05-1996
			US	6047355 A	04-04-2000
US 5140685	A	18-08-1992	DE	68913629 D1	14-04-1994
			DE	68913629 T2	16-06-1994
			EP	0357768 A1	14-03-1990
			JP	2572136 B2	16-01-1997
			JP	2501603 T	31-05-1990
			WO	8908883 A1	21-09-1989

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) **World Intellectual Property Organization**
International Bureau

(43) **International Publication Date**
5 July 2001 (05.07.2001)

PCT

(10) **International Publication Number**
WO 01/048596 A3

(54) **Title:** READ LOCK MISS CONTROL IN A MULTITHREADED ENVIRONMENT

(57) **Abstract:** Managing memory access to random access memory includes fetching a read lock memory reference request and placing the read lock memory reference request at the end of a read lock miss queue if the read lock memory reference request is requesting access to an unlocked memory location and the read lock miss queue contains at least one read lock memory reference request.

READ LOCK MISS CONTROL IN A MULTITHREADED ENVIRONMENT

BACKGROUND OF THE INVENTION

This invention relates to read lock memory references.

When a computer instruction (thread) needs access to a

5   memory location in the computer, it goes to a memory

controller. A memory reference instruction may request a

read lock on a particular memory location. The read lock

prevents other instructions from accessing that memory

location until a read unlock instruction for that memory

10   location gets granted.

SUMMARY OF THE INVENTION

According to one aspect of the invention, a method is

described of managing memory access to random access memory

includes fetching a read lock memory reference request and

15   placing the read lock memory reference request at the end

of a read lock miss queue if the read lock memory reference

request is requesting access to an unlocked memory location

- 1 -

and the read lock miss queue contains at least one read

lock memory reference request.

Other advantages will become apparent from the

following description and from the claims.

5   BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a communication system

employing a hardware-based multithreaded processor.

FIG. 2 is a block diagram of a memory controller for

latency limited operations used in the hardware-based

10   multithreaded processor.

FIG. 3 is a flowchart of the operation of a memory

controller in the hardware-based multithreaded processor.

FIG. 4 is a flowchart of the operation of a memory

controller in the hardware-based multithreaded processor.

15   DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, a communication system 10

includes a parallel, hardware-based multithreaded processor

12. The hardware-based multithreaded processor 12 is

coupled to a bus such as a peripheral component

- 2 -

interconnect (PCI) bus 14, a memory system 16, and a second

bus 18. The system 10 is especially useful for tasks that

can be broken into parallel subtasks or functions.

Specifically, a hardware-based multithreaded processor 12

5   is useful for tasks that are bandwidth oriented rather than

latency oriented. The hardware-based multithreaded

processor 12 has multiple microengines 22a-22f, each with

multiple hardware controlled threads that can be

simultaneously active and independently work on a task.

10 .   The hardware-based multithreaded processor 12 also

includes a central controller 20 that assists in loading

microcode control for other resources of the hardware-based

multithreaded processor 12 and performs other general

purpose computer type functions such as handling protocols,

15   exceptions, and extra support for packet processing where

the microengines 22a-22f pass the packets off for more

detailed processing such as in boundary conditions. In one

embodiment, the processor 20 is a Strong Arm® (Arm is a

trademark of ARM Limited, United Kingdom) based

20   architecture. The general purpose microprocessor 20 has an

operating system. Through the operating system the

processor 20 can call functions to operate on microengines

22a-22f. The processor 20 can use any supported operating

system, preferably a real time operating system. For the
core processor implemented as a Strong Arm architecture,
operating systems such as, MicrosoftNT real-time, VXWorks
and μCUS, a freeware operating system available over the

5   Internet, can be used.

The hardware-based multithreaded processor 12 also
includes a plurality of function microengines 22a-22f.
Functional microengines (microengines) 22a-22f each
maintain a plurality of program counters in hardware and

10   states associated with the program counters. Effectively,
a corresponding plurality of sets of threads can be
simultaneously active on each of the microengines 22a-22f
while only one is actually operating at any one time.

In one embodiment, there are six microengines 22a-22f

15   as shown. Each microengines 22a-22f has capabilities for
processing four hardware threads. The six microengines
22a-22f operate with shared resources including memory
system 16 and bus interfaces 24 and 28. The memory system
16 includes a Synchronous Dynamic Random Access Memory

20   (SDRAM) controller 26a and a Static Random Access Memory
(SRAM) controller 26b. SDRAM 16a and SDRAM controller 26a
are typically used for processing large volumes of data,
e.g., processing of network payloads from network packets.

- 4 -

SRAM 16b and SRAM controller 26b are used in a networking

implementation for low latency, fast access tasks, e.g.,

accessing look-up tables, memory for the core processor 20,

and so forth.

5   The six microengines 22a-22f access either the SDRAM

16a or SRAM 16b based on characteristics of the data.

Thus, low latency, low bandwidth data is stored in and

fetched from SRAM 16b, whereas higher bandwidth data for

which latency is not as important, is stored in and fetched

10   from SDRAM 16a. The microengines 22a-22f can execute

memory reference instructions to either the SDRAM

controller 26a or the SRAM controller 26b.

        Advantages of hardware multithreading can be explained

by SRAM or SDRAM memory accesses. As an example, an SRAM

15   access requested by a Thread_0, from a microengine 22a-22f

will cause the SRAM controller 26b to initiate an access to

the SRAM 16b. The SRAM controller 26b controls arbitration

for the SRAM bus 27, accesses the SRAM 16b, fetches the

data from the SRAM 16b, and returns data to the requesting

20   microengine 22a-22f. During an SRAM 16b access, if the

microengine, e.g., 22a, had only a single thread that could

operate, that microengine would be dormant until data was

returned from the SRAM 16b. The hardware context swapping

- 5 -

within each of the microengines 22a-22f enables other

contexts with unique program counters to execute in that

same microengine. Thus, another thread, e.g., Thread_1,

can function while the first thread, e.g., Thread_0, is

5   awaiting the read data to return. During execution,

Thread_1 may access the SDRAM memory 16a. While Thread_1

operates on the SDRAM unit 26a, and Thread_0 is operating

on the SRAM unit 26b, a new thread, e.g., Thread_2, can now

operate in the microengine 22a. Thread_2 can operate for a

10   certain amount of time until it needs to access memory or

perform some other long latency operation, such as making

an access to a bus interface. Therefore, simultaneously,

the processor 12 can have a bus operation, SRAM operation,

and SDRAM operation all being completed or operated upon by

15   one microengine 22a and have one more thread available to

process more work in the data path.

The hardware context swapping also synchronizes

completion of tasks. For example, two threads could hit

the same shared resource, e.g., SRAM 16b. Each one of

20   these separate functional units, e.g., the FBUS interface

28, the SRAM controller 26b, and the SDRAM controller 26a,

when they complete a requested task from one of the

microengine thread contexts reports back a flag signaling

- 6 -

completion of an operation. When the flag is received by

the microengine 22a-22f, the microengine 22a-22f can

determine which thread to turn on.

    One example of an application for the hardware-based

5   multithreaded processor 12 is as a network processor. As a

network processor, the hardware-based multithreaded

processor 12 interfaces to network devices such as a media

access controller device, e.g., a 10/100BaseT Octal MAC 13a

or a Gigabit Ethernet device 13b. In general, as a network

10   processor, the hardware-based multithreaded processor 12

can interface to any type of communication device or

interface that receives/sends large amounts of data. If

communication system 10 functions in a networking

application, it could receive a plurality of network

15   packets from the devices 13a, 13b and process those packets

in a parallel manner. With the hardware-based

multithreaded processor 12, each network packet can be

independently processed.

    Another example for use of processor 12 is a print

20   engine for a postscript processor or as a processor for a

storage subsystem, i.e., RAID disk storage. A further use

is as a matching engine. In the securities industry for

example, the advent of electronic trading requires the use

- 7 -

of electronic matching engines to match orders between

buyers and sellers. These and other parallel types of

tasks can be accomplished on the system 10.

Each of the functional units, e.g., the FBUS interface

5   28, the SRAM controller 26b, and the SDRAM controller 26a,

are coupled to one or more internal buses. The internal

buses are dual, 32-bit buses (i.e., one bus for read and

one for write). The hardware-based multithreaded processor

12 also is constructed such that the sum of the bandwidths

10   of the internal buses in the processor 12 exceeds the

bandwidth of external buses coupled to the processor 12.

The processor 12 includes an internal core processor bus

32, e.g., an ASB bus (Advanced System Bus), that couples

the processor core 20 to the memory controller 26a, 26b and

15   to an ASB translator 30. The ASB bus 32 is a subset of the

so-called AMBA bus that is used with the Strong Arm

processor core. The processor 12 also includes a private

bus 34 that couples the microengine units to SRAM

controller 26b, ASB translator 30, and FBUS interface 28.

20   A memory bus 38 couples the memory controllers 26a, 26b to

the bus interfaces 24 and 28 and memory system 16 including

a flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, the SRAM controller 26b for the

SRAM 16b is shown. The SRAM controller 26b includes an

address and command queue 120. SRAM controller 26b is

optimized based on the type of memory operation, i.e., a

5   read or a write. The address and command queue 120

includes a high priority queue 120a (holding memory

reference requests from high priority tasks), a read queue

120b (holding read memory reference requests, which are the

predominant memory reference functions that an SRAM

10   performs), and an order queue 120c (holding, in general,

all writes to SRAM 16b and reads that are to be non-

optimized). Although not shown, the address and command

queue 120 could also include a write queue. An insert

queue control unit 132 of control logic determines where to

15   queue each memory request from the microengines 22a-22f.

An arbitration scheme in a remove queue arbitration unit

124 of control logic determines the processing order of the

memory reference requests in the queues 120.

The SRAM controller 26b also includes core bus

20   interface logic, e.g., ASB bus 122. The ASB bus interface

logic 122 interfaces the core processor 20 to the SRAM

controller 26b. The ASB bus 122 is a bus that includes a

32-bit data path and a 28-bit address path. The data is

- 9 -

accessed to and from memory 16b through an MEM

(microelectromechanical) ASB data device 128, e.g., a

buffer. The MEM ASB data device 128 is a queue for write

data. If there is incoming data from the core processor 20

5   via ASB interface 122, the data can be stored in the MEM

ASB data device 128 and subsequently removed from the MEM

ASB data device 128 through an SRAM interface 140 to SRAM

memory 16b. Although not shown, the same queue structure

can be provided for reads.

10   The memory requests are sent to SRAM interface 140 via

a multiplexer 126. The multiplexer 126 is controlled by an

SRAM arbiter 131 which detects the fullness of each of the

queues and the status of the requests and from that decides

priority based on a programmable value stored in a priority

15   service control register 130. Once control to the

multiplexer 126 selects a memory reference request, the

memory reference request is sent to a command decoder and

address generator 138 where it is decoded and an address is

generated.

20   The SRAM controller 26b maintains control of the

Memory Mapped off-chip SRAM and Expansion ROM. The SRAM

controller 26b can address, e.g., 16 MBytes, with, e.g., 8

MBytes mapped for SRAM 16b, and 8 MBytes reserved for

- 10 -

special functions including: Boot space via flashrom 16c;

and Console port access for MAC devices 13a, 13b and access

to associated (RMON) counters. The SRAM is used for local

look-up tables and queue management functions.

5   The SRAM controller 26b supports the following

transactions:

Microengine requests   (via private bus) to/from SRAM.

Core Processor   (via ASB bus) to/from SRAM.

10

The SRAM controller 26b performs memory reference

sorting to minimize delays (bubbles) in the pipeline from

the SRAM interface 140 to memory 16b. The SRAM controller

26b does memory reference sorting based on the read

15   function. A bubble can either be one or two cycles

depending on the type of memory device employed.

The SRAM controller 26b includes a lock look-up device

142, an eight-entry address content addressable memory

(CAM) for look-ups of read locks. Each position includes a

20   valid bit that is examined by subsequent read lock

requests. The address and command queue 120 also includes

a read lock miss (fail) queue 120d. The read lock miss

queue 120d is used to hold up to twenty-four read requests

- 11 -

that fail because of a lock existing on the portion of
memory requested to be read. One of the microengines 22a-
22f issues a read lock memory reference request that is
processed in address and control queue 120.

5   Referring to FIG. 3, a memory reference request is
processed 51 by the controller 26b. The controller 26b
receives 50 a memory reference request on either the order
queue 120c or the read queue 120b. The command decoder and
address generator 138 determines 52 whether the memory

10   reference request is a "read" request, a write request, a
"read lock" request, or a write unlock ("unlock") request.
If the command decoder and address generator 138 determines
52 that the request is a read request, then the command
decoder and address generator 138 enters 54 the lock into

15   the lock look-up device 142 so that subsequent read lock
requests will find the memory location locked. Lock
requests cannot be processed on that locked memory location
until an unlock request from a microengine 22a-22f unlocks
58 that memory location.

20   If the command decoder and address generator 138
determines 52 that the memory reference request is a read
lock request, the controller 26b will access 56 lock look-
up device 142 to determine whether this memory location is

- 12 -

already locked. The access 56 determines whether the read

lock request gets performed 62 or enters 64 the read lock

miss queue 120d where it waits for the memory location to

be unlocked.

5   If the memory location is locked from any prior lock

request, the read lock request fails and gets stored 64 at

the end of the read lock miss queue 120d. The controller

26b proceeds to receive 50 the next request.

          If the memory location is unlocked, the lock look-up

10   device 142 checks 60 to see if the read lock miss queue

120d is empty. If the read lock miss queue 120d is empty,

that indicates that there are no previous read lock

requests awaiting processing. Thus, the read lock request

will be processed: the address of the memory reference will

15   be used by the SRAM interface 140 to perform 62 an SRAM

address read/write request to memory 16b and subsequently

lock the memory location. Then the controller 26b receives

50 the next request.

          If the read lock miss queue 120d is not empty, that

20   indicates that there is at least one prior read lock

request awaiting processing. To ensure a first-in/first-

out order of processing the read lock requests, the read

lock request will be stored 64 at the end of the read lock

- 13 -

miss queue 120d. Read lock requests continue queuing in

the read lock miss queue 120d until the controller 26b

receives 50 an unlock request and recognizes 52 it as such.

A memory location is unlocked by operation of a

5   microcontrol instruction in a program after the need for

the lock has ended. Once the unlock request is received

50, the memory location is unlocked 58 by clearing the

valid bit in the CAM 142. After the unlock 58, the read

lock fail queue 120d becomes the highest priority queue

10   120, giving all queued read lock misses a chance to issue a

memory lock request.

Referring to FIG. 4, the read lock miss queue 120d is

tested 69 to see if the head of the read lock miss queue

120d may be granted its lock. The testing 69 will continue

15   until the read lock miss queue 120d fails to provide a

grant or until there are no more entries in the read lock

miss queue 120d. The highest priority is given to the read

lock miss queue 120d by letting it win 70 the next

arbitration in the remove queue arbitration unit 124. The

20   lock look-up device 142 checks 72 to see if the top read

lock request (the one stored first in the read lock miss

queue 120d) is requesting to read an unlocked memory

location. If the read lock request is for a locked

- 14 -

location, the read lock miss queue 120d remains as it is to

ensure the first-in/first-out order described above, and

the testing 69 ends 78. If the read lock request is for an

unlocked location, the address of the memory reference will

5   be used by the SRAM interface 140 to perform 74 an SRAM

address read/write request to memory 16b. After performing

74, the lock look-up device 142 checks 76 to see if the

read lock miss queue 120d is empty. If it is empty, the

testing 69 ends 78. Otherwise, the testing 69 continues

10   with the read lock miss queue 120d winning 70 another

arbitration in the remove queue arbitration unit 124.

Other embodiments are within the scope of the

following claims.

What is claimed is:

1.   A method of managing memory access to random

access memory comprises:

fetching a read lock memory reference request; and

5   placing the read lock memory reference request at the

end of a read lock miss queue if the read lock memory

reference request is requesting access to an unlocked

memory location and the read lock miss queue contains at

least one read lock memory reference request.

10   2.   The method of claim 1 in which the random access

memory is located in a parallel, hardware-based

multithreaded processor.

3.   The method of claim 1 in which the random access

memory comprises a static random access memory.

15   4.   The method of claim 1 further comprises placing

the read lock memory reference at the end of a read lock

miss queue if the read lock memory reference request is

requesting access to a locked memory location.

5.   The method of claim 1 further comprises removing

20 the read lock memory reference requests from the read lock

miss queue in the order in which they were placed in the

read lock miss queue.

6.   The method of claim 1 in which the read lock

memory reference request is fetched from a read queue.

5   7.   The method of claim 1 in which the read lock

memory reference request is fetched from an order queue.

8.   The method of claim 1 in which the read lock miss

queue contains a number of entries equal to a number of

contexts that may execute the read lock memory reference

10   requests minus one.

9.   The method of claim 1 further comprises placing a

read lock memory reference request in a queue that contains

a read lock memory reference request requesting access to

the same memory location.

15   10.   An article comprising a computer-readable medium

which stores computer-executable instructions for managing

memory access to random access memory, the instructions

causing a computer to:

fetch a read lock memory reference request; and

20   place the read lock memory reference request at the

- 17 -

end of a read lock miss queue if the read lock memory

reference request is requesting access to an unlocked

memory location and the read lock miss queue contains at

least one read lock memory reference request.

5   11. The article of claim 10 in which the random

access memory is located in a parallel, hardware-based

multithreaded processor.

         12. The article of claim 10 in which the random

access memory comprises a static random access memory.

10   13. The article of claim 10 in which the instructions

further cause a computer to place the read lock memory

reference request at the end of a read lock miss queue if

the read lock memory reference request is requesting access

to a locked memory location.

15   14. The article of claim 10 in which the instructions

further cause a computer to remove the read lock memory

reference requests from the read lock miss queue in the

order in which they were placed in the read lock miss

queue.

15. The article of claim 10 in which the read lock memory reference request is fetched from a read queue.

16. The article of claim 10 in which the read lock memory reference request is fetched from an order queue.

17. The article of claim 10 in which the read lock miss queue contains a number of entries equal to a number of contexts that may execute the read lock memory reference requests minus one.

18. The article of claim 10 in which the instructions further cause a computer to place a read lock memory reference request in a queue that contains a read lock memory reference request requesting access to the same memory location.

19. A controller for managing access to random access memory comprises:

an address content addressable memory that supports locked locations in memory; and

a read lock miss queue that stores read lock memory reference requests that request access to memory locations unlocked in the address content addressable memory when the read lock miss queue contains at least one entry.

20. The controller of claim 19 in which the random access memory is located in a parallel, hardware-based multithreaded processor.

21. The controller of claim 19 in which the random access memory comprises a static random access memory.

22. The controller of claim 19 in which the read lock memory reference requests are stored at the end of the read lock miss queue if the read lock memory reference request requests access to a locked memory location.

23. The controller of claim 19 in which the read lock memory reference requests are removed from the read lock miss queue in the order in which they were placed in the read lock miss queue.

24. The controller of claim 19 in which the read lock miss queue contains a number of entries equal to a number of contexts that may execute the read lock memory reference requests minus one.
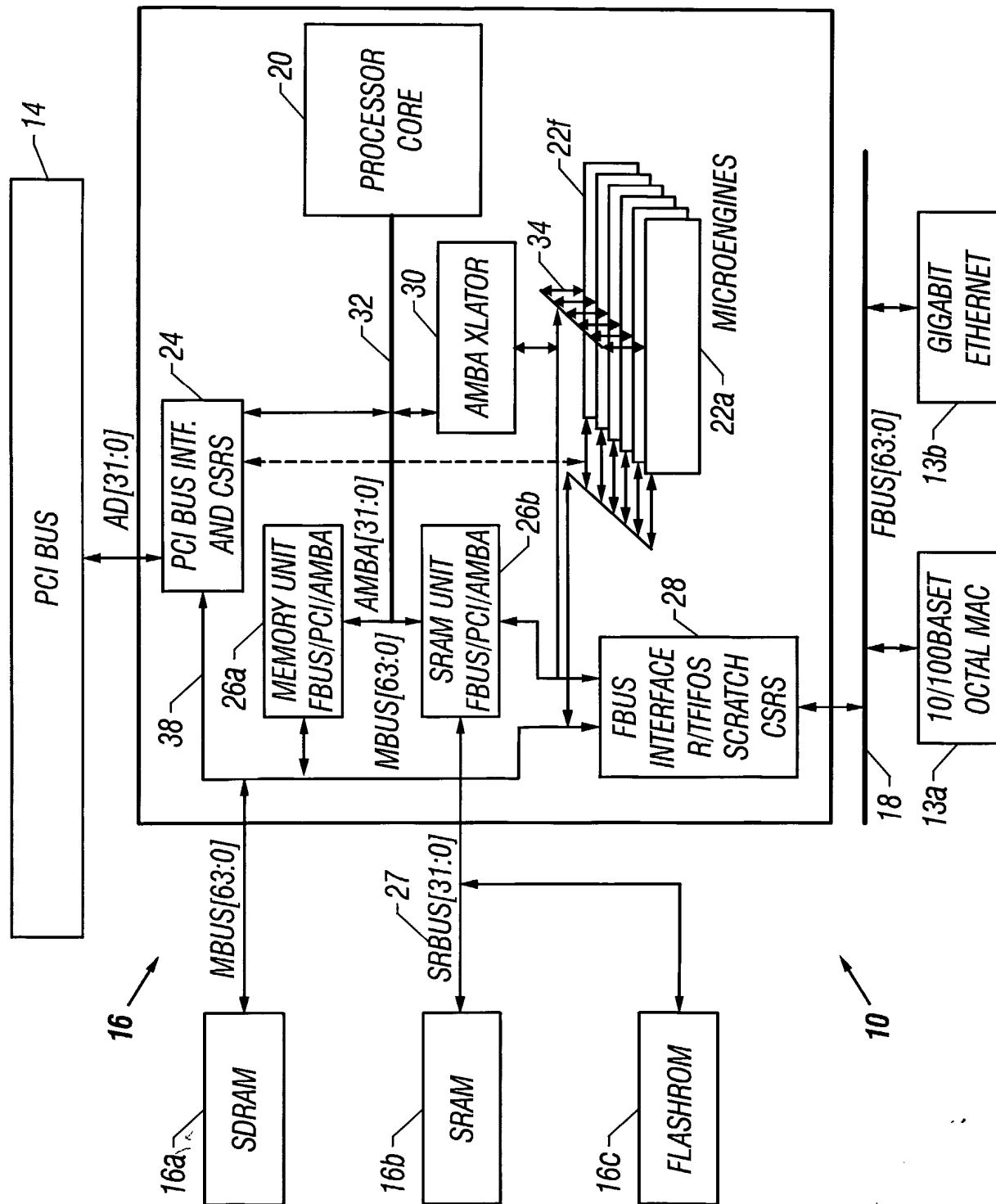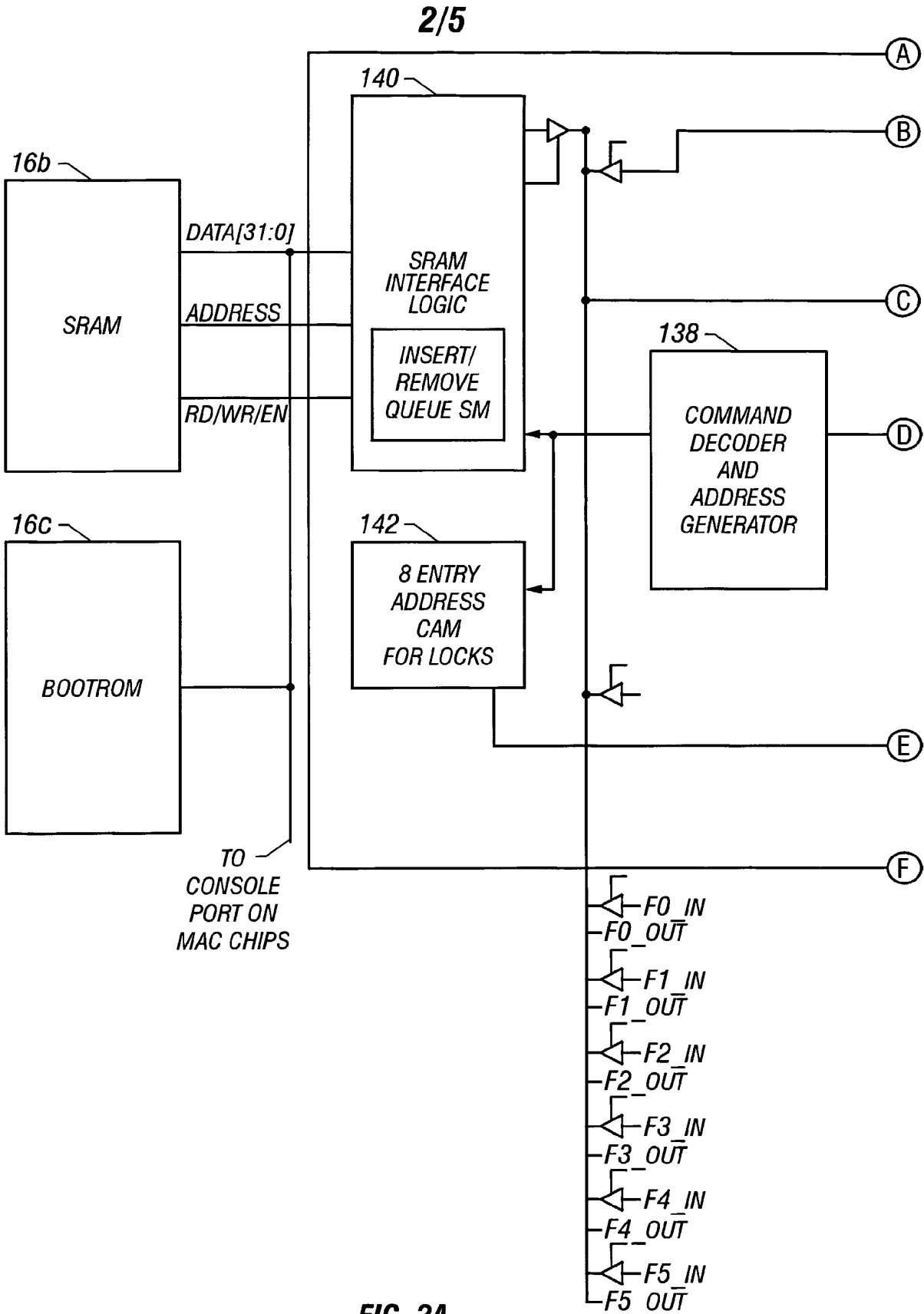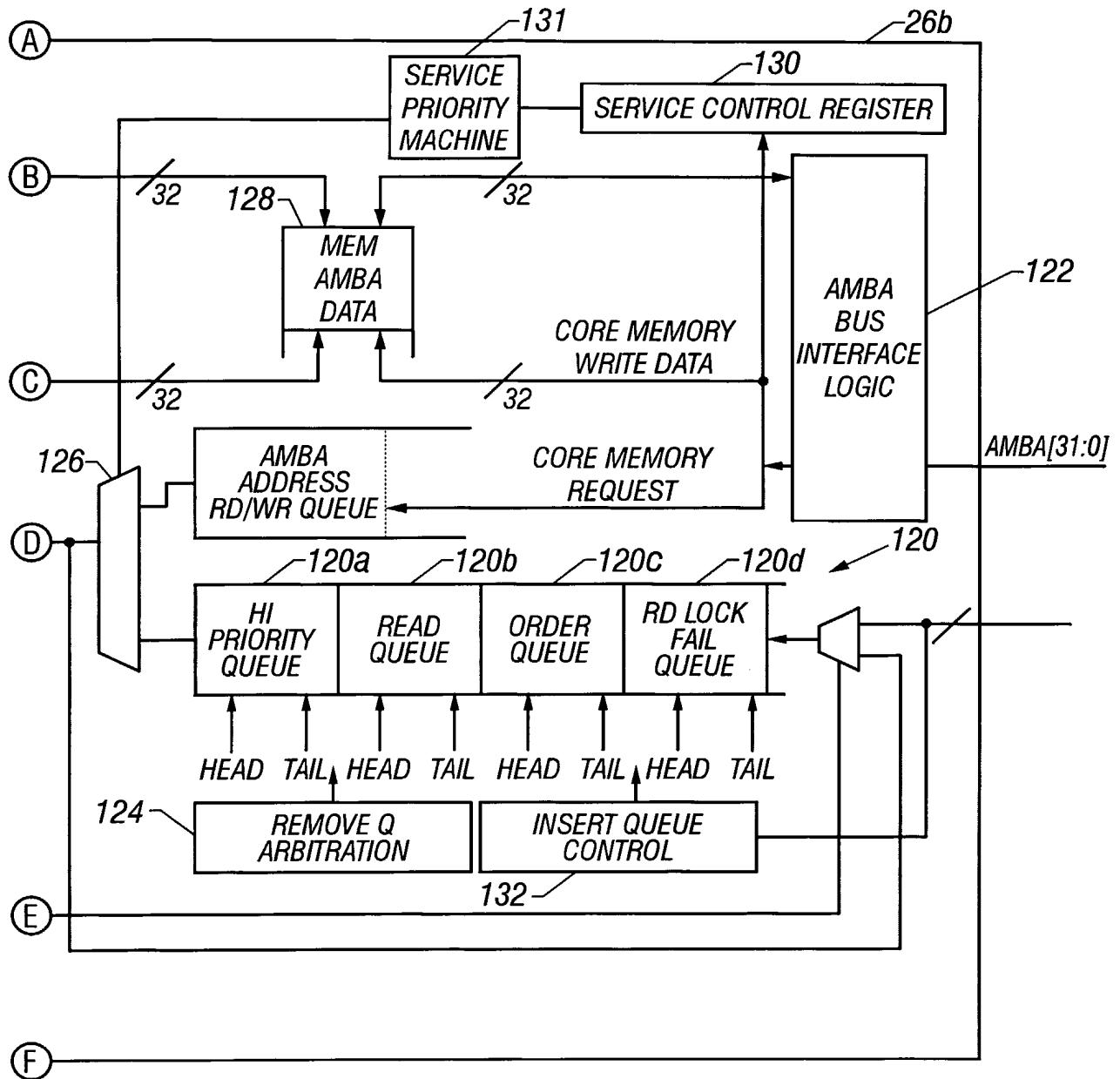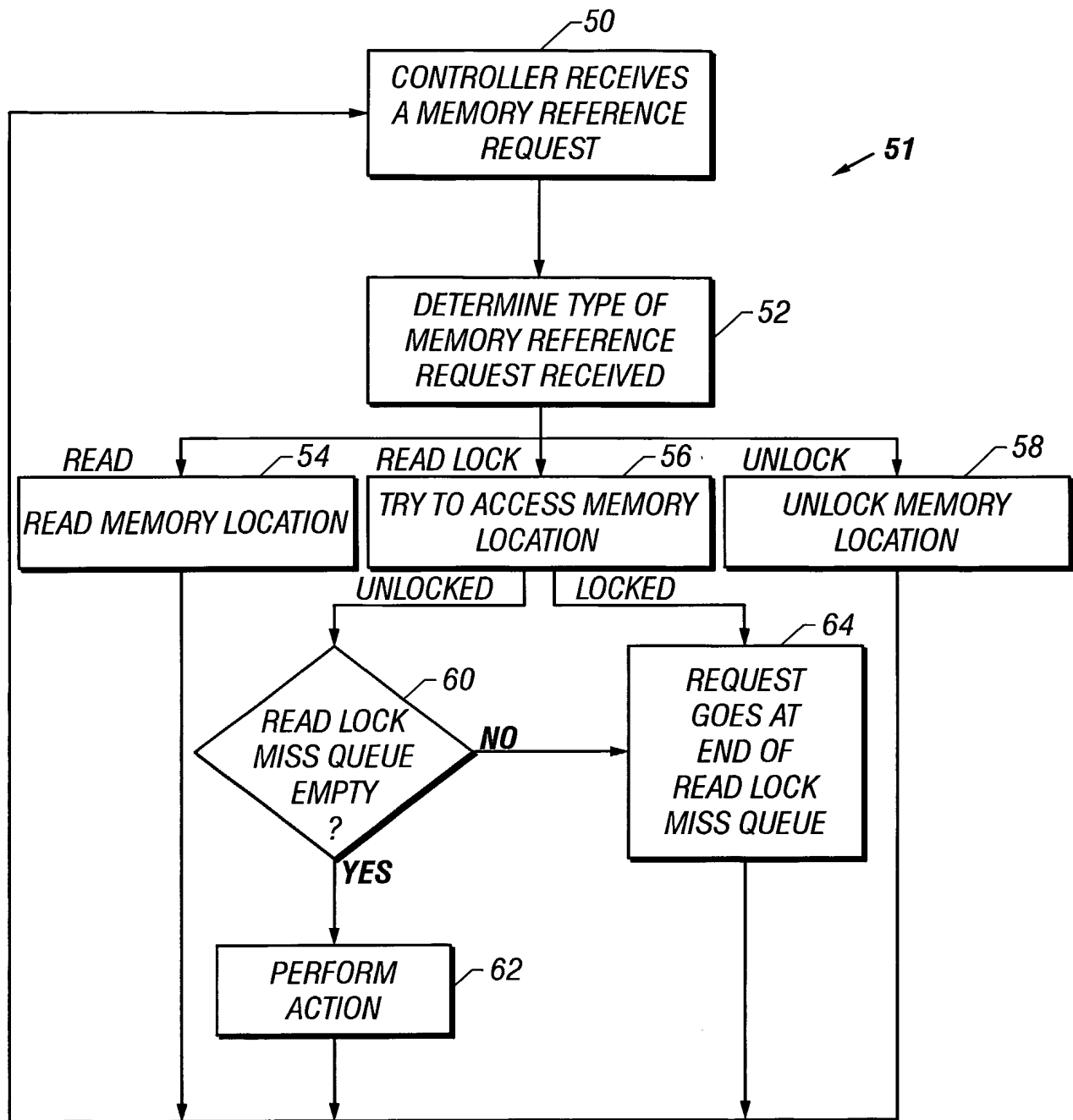
FIG. 1

FIG. 2A

FIG. 2B

FIG. 3

5/5

FIG. 4

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7   G06F9/52

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7   G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 464 715 A (DIGITAL EQUIPMENT CORP) 8 January 1992 (1992-01-08) column 7, line 31 – line 42 column 10, line 55 –column 11, line 39 column 12, line 7 –column 13, line 40 column 14, line 6 – line 41 –––	1-24
A	US 5 517 648 A (SANFACON MARC ET AL) 14 May 1996 (1996-05-14) column 11, line 21 – line 53 column 14, line 25 – line 41 column 18, line 14 –column 19, line 15 ––– -/--	2,3,6,7, 11,12, 15,16, 20,21

| X | Further documents are listed in the continuation of box C. | | X | Patent family members are listed in annex. |

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 November 2001	16/11/2001

| Name and mailing address of the ISA
European Patent Office, P.B. 5818 Patentlaan 2
NL – 2280 HV Rijswijk
Tel. (+31–70) 340–2040, Tx. 31 651 epo nl,
Fax: (+31–70) 340–3016 | Authorized officer

Carciofi, A |

International Application No

PCT/US 00/33395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 140 685 A (SIPPLE RALPH E ET AL) 18 August 1992 (1992-08-18) column 4, line 54 –column 5, line 40 column 13, line 32 –column 14, line 2 ─────	1,4,5, 10,13, 14,19, 22,23

INTERNATIONAL SEARCH REPORT

Information on patent family members

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0464715	A	08-01-1992	AU	633724 B2	04-02-1993
			AU	7527991 A	12-03-1992
			CA	2042772 A1	30-12-1991
			DE	69128815 D1	05-03-1998
			DE	69128815 T2	27-08-1998
			EP	0464715 A2	08-01-1992
			US	5341491 A	23-08-1994
US 5517648	A	14-05-1996	US	6311286 B1	30-10-2001
			US	5956522 A	21-09-1999
			US	6125436 A	26-09-2000
			US	5809340 A	15-09-1998
			US	5522069 A	28-05-1996
			US	6047355 A	04-04-2000
US 5140685	A	18-08-1992	DE	68913629 D1	14-04-1994
			DE	68913629 T2	16-06-1994
			EP	0357768 A1	14-03-1990
			JP	2572136 B2	16-01-1997
			JP	2501603 T	31-05-1990
			WO	8908883 A1	21-09-1989